Docket No.: GR 98 P 8110

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applic. No.	:	09/816,933
Inventor	:	Christian Siemers
Filed	:	March 23, 2001
Title	:	Program-Controlled Unit

DECLARATION under 37 C.F.R. § 1.131

I, the undersigned Peter Jannig, hereby declare:

I have first-hand knowledge that the invention of the above-identified application was made in a WTO country prior to September 14, 1998.

I am a German patent attorney (*Patentanwalt*) and I am a person designated in 37 C.F.R. § 1.56(c), who is an attorney/agent who prepared the application and who was substantively involved in the preparation and prosecution of the application; I prepared the application text which was filed in the German Patent Office (DE 198 43 637) on September 23, 1998 and which later became the above-identified patent application substantially exclusively from the description of the invention given to me by the corporate patent department of Siemens AG on July 9, 1998.

It is my firm belief, as a patent professional and a trained engineer, that the inventor Dr. Siemers was in the complete possession of the invention prior to September 14, 1998. Further, I diligently worked on the completion of the German application text from just prior to September 14, 1998 and leading up to the filing of September 23, 1998.

I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that I have been warned that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001, and such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Peter Jannig

Peter Jannig

August 7, 2006

Date

Docket No.: GR 98 P 8110

IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applic. No. : 09/816,933
Inventor : Christian Siemers
Filed : March 23, 2001
Title : Program-Controlled Unit

DECLARATION under 37 C.F.R. § 1.131

I, the undersigned sole inventor hereby declare:

The invention of the above-identified application was made in a WTO country prior to September 14, 1998.

I wrote and then submitted in early July 1998 to the patent department of Siemens AG (attention Mr. Hassa), assignee of the subject invention, a substantially complete description of the invention entitled, in a first part, "A Research Concept for Hardware/Software Co-Design" and, in a second part, "Universal Configurable Machine (UCM) in Embedded Control Applications." The description of the invention was a substantially complete disclosure of the invention and it contained all of the details of the invention as it was later described and claimed in the German patent application DE 198 43 637, of September 23, 1998, and in the above-identified patent application.

Enclosed herewith, as corroborating evidence, are copies of the afore-mentioned descriptions as they were submitted in early July 1998 and verified translations thereof. The descriptions show that I was in complete possession of the invention prior to September 14, 1998.

I further declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that I have been warned that willful false statements and the like so made are punishable by fine or imprisonment, or both, under 18 U.S.C. § 1001, and such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Christian Siemers — Christian Siemers

August 7th, 2006 — Date

GR98P8110
Application No. 09/816,933

CERTIFICATION

I, the below named translator, hereby declare that: my name and post office address are as stated below; that I am knowledgeable in the English and German languages, and that I believe that the attached text is

- a true and complete translation of a paper entitled "Universal Configurable Machine (UCM) in Embedded Control Applikationen" (Universal Configurable Machine (UCM) in Embedded Control Applications), and
- a true and partial translation of a paper entitled "Ein Forschungskonzept zum Hardware/Software Co-Design" (A research concept for hardware/software co-design) by Sybille Roth, Christian Siemers.

I hereby declare that all statements made herein of my own knowledge are true and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both, under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of the application or any patent issued thereon.

Hollywood, Florida

Birgit Bartell
August 7, 2006
Lerner Greenberg Stemer LLP
P.O. Box 2480
Hollywood, FL 33022-2480
Tel.: (954) 925-1100
Fax.: (954) 925-1101

Universal Configurable Machine (UCM) in Embedded Control Applications

Applications in the embedded control area are primarily event-controlled. This is due to the fact that, because the controller is embedded in the higher-ranking machine, the non-deterministic sequence of events mandates a special program form for event handling instead of algorithmic-deterministic programming.

Normally these external interruptions are handled by interrupt service routines within the overall application, i.e. embedded in real-time operating systems. A real-time operating system contains a special part for this purpose, the task management system (TMS), which contains an anomaly compared with the other basic parts of an operating system.

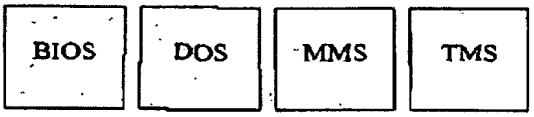

Fig. 1: Columns of an operating system

The basic input output system (BIOS), disk operating system (DOS) and memory management system (MMS) give the user a higher functionality than the simple processor; they form a virtual machine. TMS (in real-time operating systems), on the other hand, manages the computing time deficit; i.e., above all the computing capacity of the processor is distributed in such a way that the requirements of the outside world are mirrored in the available computing time in accordance with the specifications.

The system designer, however, has only an indirect influence on the real-time functionality in that he has to rely on the operating system and its specifications. Especially the verification of the application with respect to the response being correct in terms of time (and mathematically) in all circumstances under hard real-time conditions is an extremely intensive process, and in some cases even impossible. This is where USM architecture (Fig. 2) comes into play:

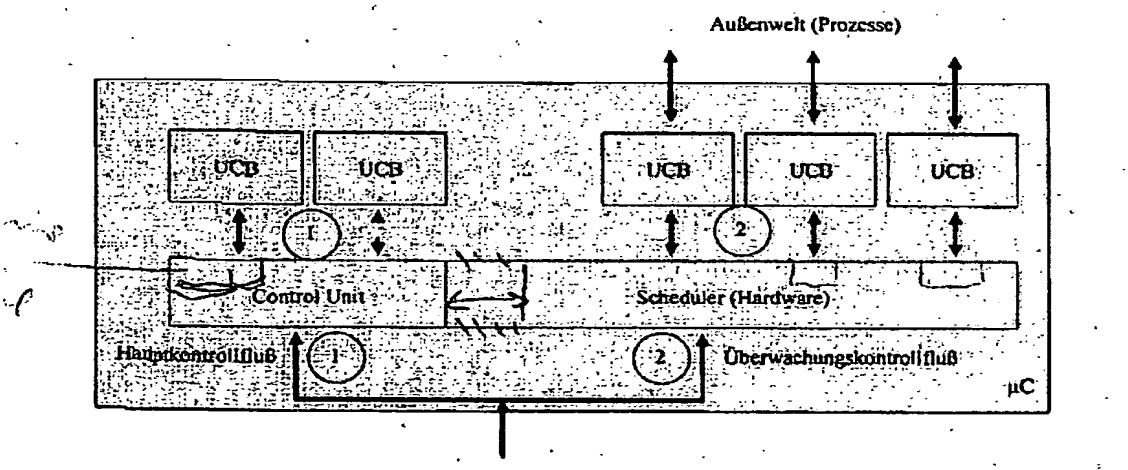

Abb. 2: Blockstruktur UCM

Fig. 2: Block Structure UCM

USM architecture consists of two main components. The control unit, essentially the same as in previous microcontrollers, controls the processing of the main program, i.e. the actual application. For this purpose it uses one or more universal configurable blocks

(UCB), which are comparable to reconfigurable arithmetical logical units and which process commands in blocks.

The second part is very important for the new type of processing in controllers and consists of (small) UCBs and a scheduler integrated in hardware. This scheduler is responsible for allocating computing time and monitoring control flow, which exists in so-called threads (*Befehlsfäden*). This architecture gives UCM architecture the following characteristics:

- It is a single processor multiblock system rather than a multiprocessor system. A great advantage of this is that it is not necessary to perform intensive processor synchronizations, but merely to have a central unit as a distribution entity for computing time demand.

  The multiblock system offers parallel computing capacity and is optimized for threads. Threads form the optimal basis for system integration in the context of interruption processing, which – as explained – [is] essential to embedded control applications.

- The developer is able to build a deterministic application in the real-time domain as well with the aid of UCM architecture. This is possible through the explicit, possibly even exclusive, allocation of UCBs to threads for high-priority interrupts, without any load being placed on the core CPU. The operating system block for task management, whose deterministic behavior is so difficult to estimate, is either completely replaced by hardware or restricted to a few critical paths.

- UCBs are based on the principle of structural programming (field programmable logic similar to FPGAs) and are therefore substantially faster compared with sequential processing using processors. This results in a dramatic acceleration of the thread processing to the application's benefit.

  On the other hand, some studies have already shown that the program environment hardly has to be changed for UCBs compared with normal CPUs. For the software developer, the programming language remains the same, and known compiler technologies can be used, particularly for optimization. Upgrades remain highly limited, they can be integrated in hardware (controller) or in software, and they remain hidden to the developer.

- The multiblock design of UCM architecture additionally makes possible an error tolerance with respect to component failure, because instruction blocks can be routed from the scheduler and the control unit to a functioning UCB as needed.

Conclusion: UCM architecture makes it possible to design controllable deterministic applications even in the complex field of real-time applications. There is greater consistency between the description and the real machine because the part of the virtual machine operating system TMS which has to be considered one of the least deterministic is replaced by the real machine. UCM increases the controllability of embedded control applications while exhibiting tolerance to component failure.

Übersicht + Zusammenf.
97E8176, 97E8177, 98E8084 Hassa
03. JULI 1998

Universal Configurable Machine (UCM) in Embedded Control Applikationen

Applikationen im Embedded-Control-Bereich sind überwiegend Ereignis-gesteuert. Der Grund hierfür ist in der Einbettung des Controllers in die übergeordnete Maschine zu sehen, die durch die nicht-determinstische Folge von Ereignissen keine algorithmisch-deterministische Programmierung ermöglicht, sondern eine spezielle Programmform zur Bedienung der Ereignisse erzwingt.

Üblicherweise werden diese externen Unterbrechungen durch Interrupt-Service-Routinen innerhalb der Gesamtapplikation bzw. eingebettet in Echtzeit-Betriebssysteme bedient. Ein Echtzeit-Betriebssystem enthält für diesen Zweck einen besonders ausgeprägten Teil, das Task Management System (TMS), das verglichen mit den anderen Basisteilen eines Betriebssystem eine Besonderheit enthält.

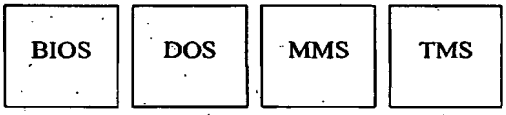

Abb. 1: Säulen eines Betriebssystems

Basic Input Output System (BIOS), Disk Operating System (DOS) und Memory Management System (MMS) stellen dem Anwender eine gegenüber dem einfachen Prozessor höhere Funktionalität zur Verfügung, sie bilden eine virtuelle Maschine. TMS (in Echtzeit-Betriebssystemen) hingegen verwaltet den Mangel an Rechenzeit, d.h., es wird vor allem die Rechenkapazität des Prozessors so verteilt, daß die Anforderungen der Außenwelt den Spezifikationen entsprechend auf die zur Verfügung stehende Rechenzeit abgebildet werden.

Der Systemdesigner aber hat nur mittelbaren Einfluß auf die Echtzeitfunktionalität, indem er sich auf das Betriebssystem und dessen Spezifikationen verlassen muß. Insbesondere die Verifikation der Applikation, bei harten Echtzeitbedingungen unter allen Umständen zeitlich (und rechnerisch) korrekt zu reagieren, ist sehr aufwendig, ggf. sogar unmöglich. Hier setzt die UCM-Architektur (Abb. 2) an:

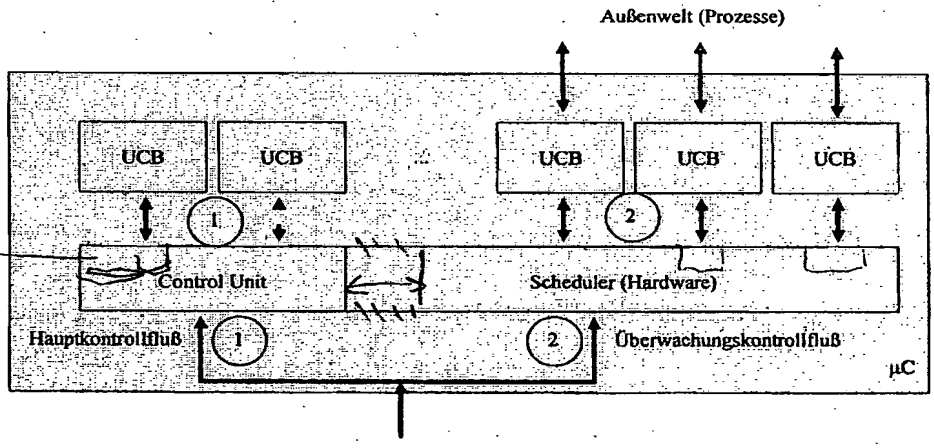

Abb. 2: Blockstruktur UCM

Die UCM-Architektur besteht aus zwei Hauptkomponenten. Die Control Unit, im wesentlichen unverändert gegenüber den bisherigen Mikrocontrollern, steuert die Bearbeitung des Hauptprogramms, also der eigentlichen Applikation. Sie nutzt hierzu ein oder mehrere Universal Configurable Blocks (UCB), die ihrerseits mit einer rekonfigurierbaren Arithmetical Logical Unit vergleichbar sind und die Befehle blockweise abarbeiten.

Der zweite Teil ist sehr wichtig für die neue Form der Bearbeitung in Controllern und besteht aus (kleinen) UCBs sowie einem in Hardware integrierten Scheduler. Dieser Scheduler ist verantwortlich für die Zuteilung von Rechenzeit und Überwachungskontrollfluß, der in sogenannten Threads (Befehlsfäden) vorliegt. Durch diese Architektur ergeben sich folgende Charakteristika für die UCM-Architektur:

- Es handelt sich nicht um ein Multiprozessorsystem, sondern um ein Einprozessor-Multiblocksystem. Dies hat den großen Vorteil, keine aufwendigen Prozessorsynchronisationen durchführen zu müssen, sondern eine zentrale Einheit als Verteilungsinstanz für Rechenzeitbedarf zu besitzen.

Das Multiblocksystem bietet parallele Rechenkapazität an und ist auf Threads optimiert. Threads bilden im Rahmen der Unterbrechungsbearbeitung, die – wie bereits dargestellt – für Embedded Controlapplikationen sehr wesentlich sind, die optimale Grundlage zur Systemintegration.

- Der Entwickler ist in der Lage, mit Hilfe der UCM-Architektur eine deterministische Applikation auch im Echtzeitbereich aufzubauen. Dies erfolgt durch die explizite, ggf. sogar exklusive Zuordnung von UCBs zu Threads bei hochpriorisierten Interrupts, ohne daß hierdurch die Kern-CPU belastet wird. Der in seiner Deterministik schwierig abzuschätzende Betriebssystem-Block für Task Management wird entweder vollständig durch die Hardware ersetzt oder auf weniger kritische Pfade eingeschränkt.
- Die UCBs basieren auf einem Prinzip der strukturalen Programmierung (Feldprogrammierbare Logik ähnlich zu FPGAs) und sind dementsprechend wesentlich schneller verglichen mit sequentieller Bearbeitung durch Prozessoren. Dies resultiert in einer drastischen Beschleunigung der Thread-Bearbeitung zum Vorteil der Applikation.

  Andererseits wurde durch einige Arbeiten bereits gezeigt, daß eine Änderung der Programmierumgebung für UCBs im Vergleich zu normalen CPUs kaum erforderlich ist. Für den Softwareentwickler bleibt die Programmiersprache erhalten, bekannte Compilertechnologien insbesondere zur Optimierung können genutzt werden. Die Erweiterungen bleiben eng begrenzt, können sowohl in Hardware (Controller) als auch in Software integriert werden und bleiben für den Entwickler verborgen.
- Das Multiblock-Design der UCM-Architektur ermöglicht zusätzlich eine Fehlertoleranz gegen Ausfall von Komponenten, indem die Instruktionsblöcke von dem Scheduler und der Control Unit entsprechend zu einem funktionsfähigen UCB geroutet werden können.

Fazit: Die UCM-Architektur ermöglicht, auch im schwierigen Feld der Echtzeitanwendungen beherrschbare, deterministische Applikationen zu erstellen. Sie erhöht die Konsistenz zwischen Beschreibung und realer Maschine, indem der Teil der virtuellen Maschine Betriebssystem TMS, der als wenig deterministisch betrachtet werden muß, durch die reale Maschine ersetzt wird. UCM erhöht die Beherrschbarkeit von Embedded Control Applikationen bei gleichzeitiger Toleranz gegenüber Ausfall von Komponenten.

Partial Translation of a Paper Entitled

Ein Forschungskonzept zum
Hardware/Software Co-Design
(A Research Concept for
Hardware/Software Co-Design)
by
Sybille Roth, Christian Siemers

[Translator's note: Translation starting p. 45, line 16]

5.1 The ASSC Architecture on the Block Level

Fig. 5.2 shows the general structure of ASSC architecture. In this figure, the ROM and RAM memory areas are represented as external memories; this should be considered merely exemplary since integrated architectures can also be designed for a comparable expenditure, in part even a lower one.

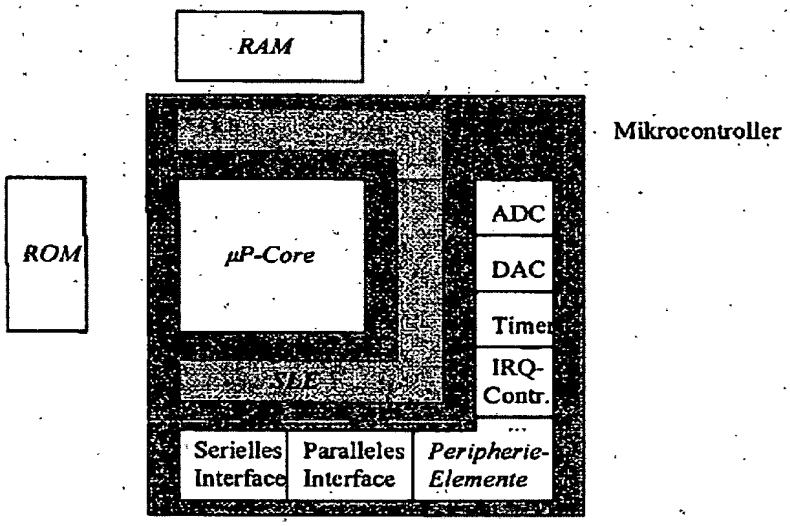

Abb. 5-2: Blockstruktur ASSC-Architektur

In contrast with a traditional microcontroller architecture, in ASSC architecture there is a layer between peripheral elements and microprocessors, which consists of structurable logic elements, hereafter SLE. This SLE layer contains permanent connections between the microprocessor core and the peripherals in order to create functional compatibility with previous architectures, as well as configurable data paths and data links whose structures will be presented in detail in the next section and whose coupling with the microprocessor will be presented in the section following the next section.

The SLE layer can be built in the area of the data memory in two variants:

- Variant 1 does not contain a direct interface to the data memory. This basically prevents competing access (core and periphery via SLE) to the data memory. The SLE (and via this, the peripheral elements) can access the memory indirectly by means of injected processor commands.

- Variant 2 contains a direct interface to the data memory (dotted lines in Fig. 5-2) and therefore must also offer an expanded bus interface in order to be able to resolve competing accesses. The advantage of this solution consists in the significantly faster handling of memory transfer, although it requires a larger outlay.

In both variants the SLE layer otherwise has the same attributes and forms an intelligent, configurable interface between the periphery and the core and the peripheral elements.

5.2 The Fine Structure of the Structurable Logic Elements

Generally a unified block structure in the SLE layer is possible. This consists of one or two (successive) logic blocks for disjunctive normal form (or other base systems), so (almost) any combination of input signals can be generated.

This architecture is highly flexible but creates a large internal wiring complexity because each input and output has to be led to the logic gates twice. In DNF architecture this leads to AND gates with a large number of inputs. For this reason, in this paper an additional sub-block structure is proposed, which reduces the complexity of the gates; this hardly diminishes the flexibility of ASSC architecture given the right splitting and cross-connections among blocks.

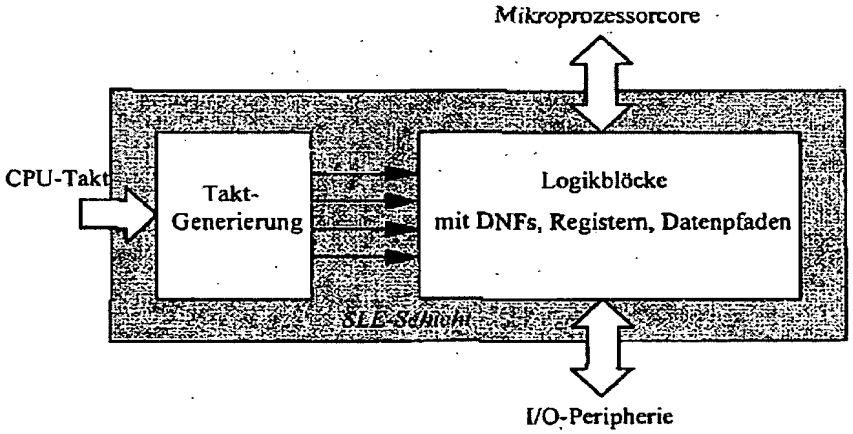

Abb. 5-3: Interne Struktur der SLE-Schicht

The fine structure of the SLE layer initially comprises two internal blocks, represented in figure 5-3. The much smaller block serves for supplying the structurable hardware with various clock signals which derive from a CPU master clock signal. For instance, clock signals with an altered frequency, altered duty ratio and/or altered phase angle are generated in this subunit.

The much larger part of the SLE layer, the actual logic blocks, contain data paths which are configurable with the aid of multiplexers, registers, and a structurable logic

arrangement in the form of disjunctive normal forms ; the multiplexers and structurable logic can be combined. Here, individual state machines with corresponding decoders are implemented, which determine the coupling between the core and periphery and the periphery/periphery and periphery/memory coupling.

5.2.1 Subunit for Clock Generation

The subunit for clock generation consist of a high-speed configurable logic arrangement, for instance in the form of a disjunctive normal form (NICHT-UND-ODER logic) with a downstream configurable memory including inversion/buffering and feedback. As a clock signal input, this subunit contains one or more CPU clock signals and k number of outputs, all of which can be utilized as a clock signal inside the logic block subunit. Figure 5-4 shows the design on the gate level; this representation is based on the diagrammatic style normally used to represent PALs/GALs and contains the basic wiring of one internal signal and one generated output signal in each case.

The DNF design (with configurable inverting) allows the generation of a number of clock signal variations; internal register/DNFs which are fed back also provide for variations with non-binary duty and division ratios.

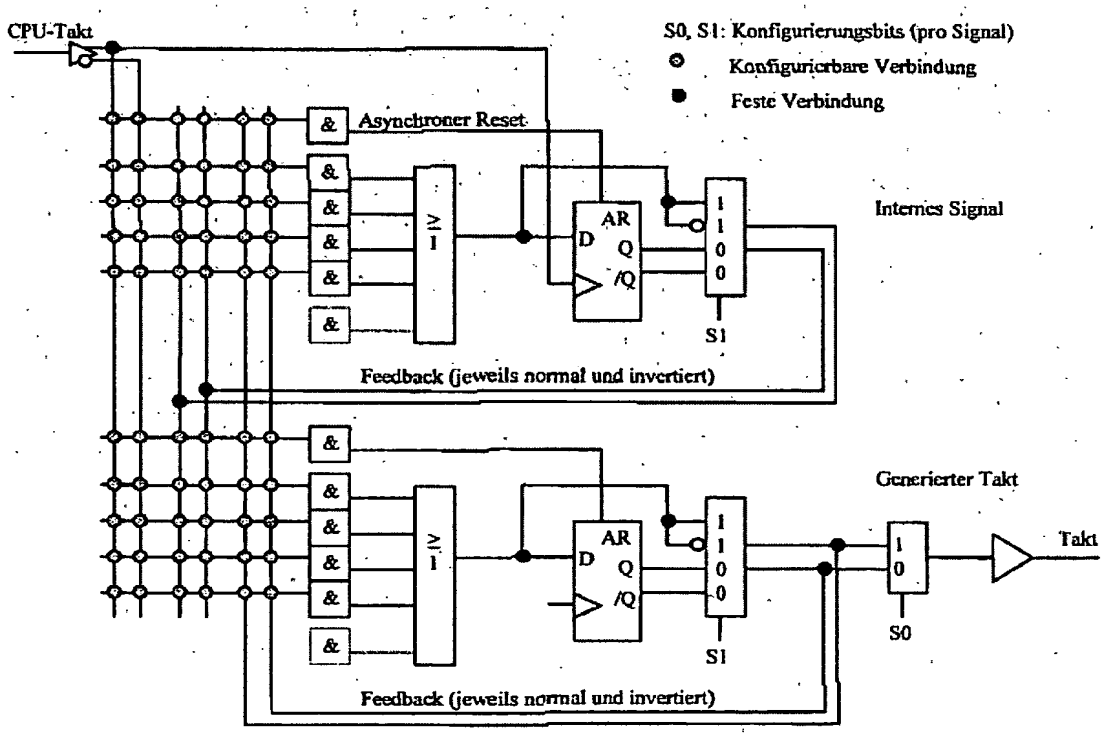

Abb. 5-4: Aufbau Taktgenerierungseinheit auf Gatterebene

The number of different clock signals that can be generated should be no less than 4, whereas an upper limit is imposed by available resources. The lower limit for the number of terms per DNF is also 4. An internal signal should additionally be available for every clock signal generated.

5.2.2 The Logic Blocks Subunit

As already explained, the logic blocks subunit consists of a multiplicity of components. Among them are:

1. Configurable multiplexer for connecting data paths; these multiplexers may coincide with the structurable hardware cited below in number 4 under certain circumstances

2. Registers with the corresponding bit width for data buffering

3. Bit registers for the coding of states

4. Structurable hardware for the decoding of states for control signals

5. Fixed and structurable hardware for linking data with other data and with constants

These components are linked with one or more logic blocks as represented in the following Fig. 5-5.

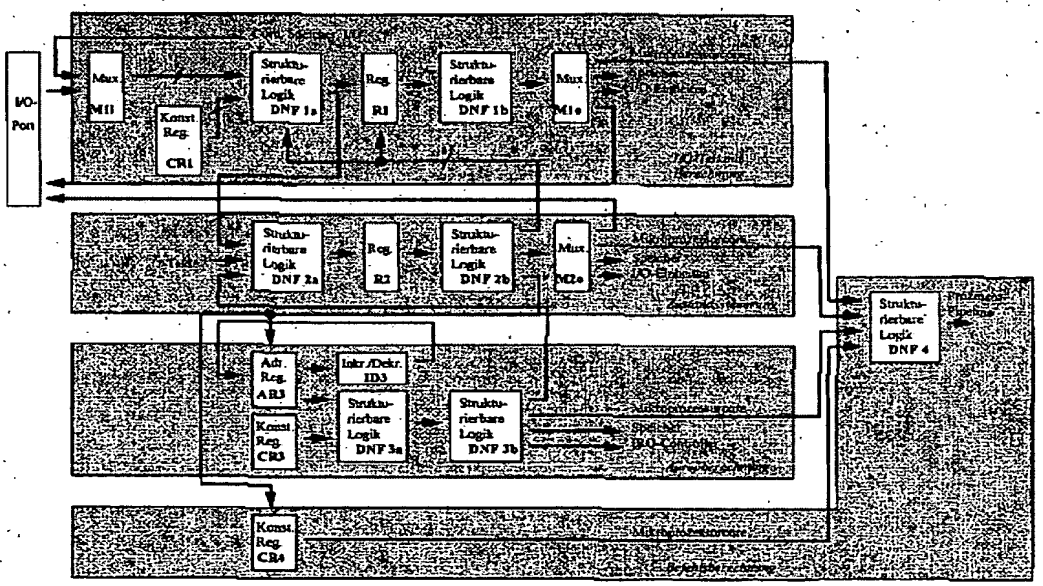

Abb. 5-5: Darstellung eines Logikblocks als Teileinheit in der SLE-Schicht

In the I/O part of a logic block, the data can be linked with each other; this is possible in configurable or permanent hardware, in Fig. 5-5 the links are integrated into DNF1a and 1b. The memory function in R1 serves for buffering, whereas the constant register CR1 serves for the storage of calculation constants (e.g. for minimum/ maximum limits).

The multiplexer M1i (source selection) and M1o (destination selection) connect the logic block with data buses to the core, memory (if allowed, e.g. in variant 2) and I/O elements. The constant k refers to the system's inherent data bus width, e.g. 16 bits. The multiplexer M1i is key to a flexible coupling between blocks; a common calculation can be performed in several blocks at once for one I/O component if the input multiplexer is able to switch the corresponding input buses. This technique creates the capability for either the parallel or serial use of calculation options.

The functionality integrated in subblock 1 serves for data flow; this subunit is comparable to the ALU of a von Neumann CPU, so the following discussion covers the principal differences between a processor and an SLE layer.

The state control requires a separate logic part, equipped with DNF 2a, 2b, registers R2 and multiplexer M2 for destination selection. Various machine types (up to Mealy machines) may be integrated in the logic part, even with different clocks as well. Generally, the number of states will be small, so there is a sufficient number with (s=) 5 bit registers (32 states). DNF 2b serves for the decoding of states for control purposes.

Subunit 2 is operatively comparable with the arithmetic logic unit (ALU) of a program-controlled unit working according to the Von Neumann Principle, of course without the command processing; the states are passed through when triggered by external signals.

The address calculation is optional and is used especially for data transfer in blocks. For this purpose, an address increment/decrement and a coupling to the sequence control are integrated. The coupling to the microprocessor core and the memory should indicate that in this case the source and destination address have to be given. Here too the memory coupling in variant 1 is dispensed with (no direct memory access). CR3 contains the final address; in AR3 the current address is stored. The partition of the logic into DNF3a and DNF3b should indicate that this is a two-stage DNF design, since the comparison logic, for example, is very complex for single-stage linkage.

Lastly, the command computation serves for the injection of microprocessor commands into the pipeline of the core. In CR4 the corresponding command code is stored in order to be able to transmit it along with address and data information. The command injection is controlled by means of signals from the state control. The connections to the microprocessor core represented on the multiplexers are combined into one interface DNF4 for this purpose, which injects a command under the control of the state machine.

The structurable logic can be integrated in any form; NAND arrays, multiplexer based variants, and a logic in a disjunctive normal form like the one chosen in this example are all possible. Such a DNF based logic arrangement is represented in Fig. 5-6 (see also clock subunit, a variation of the diagram below):

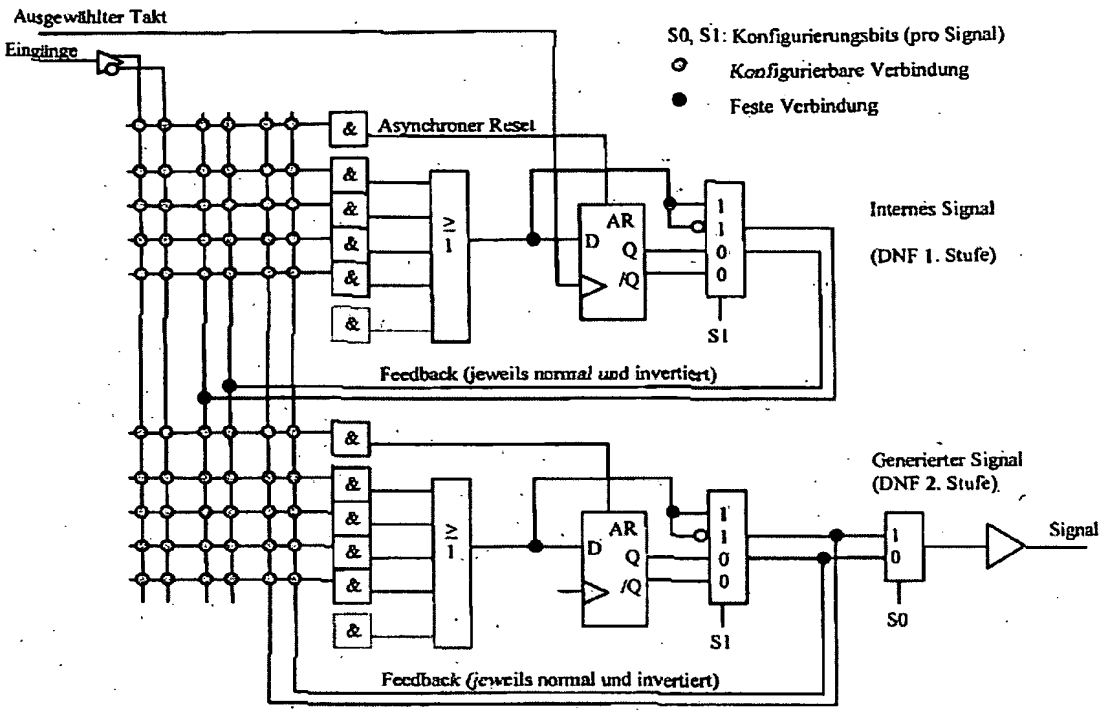

Abb. 5-6: Allgemeine Struktur der DNF in Abb. 5-5

The form of DNF represented here makes two-stage logic arrangements possible, for instance the combination of DNF2a, R2, and DNF2b with additional output registers; at any time this can be reduced to a single-stage logic by leaving out the 2nd stage and integrating the output multiplexer in stage 1. The scale of the respective logic structure must be defined by the system designer; this typifies the flexibility and power of the ASSC architecture.

In closing, it should be noted that, for data group operations, additional storage units must be integrated in the SLE. In this case, past data must also be factored into the computation, the storage of which should be done not in the memory but in the SLE part itself for reasons of performance.

. . . .

[translator's note: text continues with paragraph 5.3]

Übersicht + Zusammenf.
97E8176, 97E8177, 98E8084

Dipl.-Inf. Sybille Roth, Prof. Dr. Christian Siemers

Ein Forschungskonzept zum Hardware/Software Co-Design

Abstract: In diesem Konzeptionspaper wird ein grundlegendes Modell für Hardware/Software Co-Design vorgestellt, das auf einer universellen Hardwarearchitektur und einem Ansatz zu einer Systembeschreibungssprache basiert. Die Hardwarearchitektur, als Universal Configurable Block (UCB) bezeichnet, kann einerseits direkt struktural programmiert werden; andererseits wird in diesem Paper ein Algorithmus präsentiert, um prozedurale Programme ebenfalls in einem UCB zum Ablauf zu bringen, so daß diese Blöcke als innerste Bestandteile einer CPU geeignet sind. Die Systembeschreibungssprache wird definiert und bietet auf Threadebene eine ideale Plattform zur Konfigurierung von UCBs und zur Compilierung in prozeduralen Programme. Die endgültige Architektur, die zur Ausführung derartiger Co-Design-Programme geeignet ist, wird als Universal Configurable Machine (UCM) bezeichnet und besteht aus einer Anzahl von UCBs mit ergänzender Hardware.

1 Einleitung

Rekonfigurierbare Hardware, häufig als Feldprogrammierbare Logikbausteine (FPL) bezeichnet, ist in der Bekanntheit deutlich gestiegen: War die strukturierbare Hardware vor einiger Zeit etwas für Spezialisten, die sich mit dem Ersatz von 'fertigen' Bausteinen befaßten und sogenannte Glue Logic bzw. State Machines einfacher bis mittlerer Komplexität darin implementierten, so hat sich die Gruppe der Forscher und Entwickler in Richtung Informatik erweitert. Die Gründe für diesen neuerlichen Attraktivitätsgewinn liegen in der wachsenden Größe von feldprogrammierbaren Bausteinen, die nunmehr zur Implementierung ganzer Systeme bzw. Algorithmen in Hardware geeignet sind, bei gleichzeitig sinkenden Preisen.

Damit hat sich auch der Charakter von Hardware-Applikationen deutlich gewandelt. Rekonfigurierbare Hardware ist sehr schnell, im Vergleich zur Software ähnlich programmierbar, wenn auch im Detail deutliche Unterschiede bestehen. Es ist jedoch das Denken in ganzen Algorithmen bzw. in wesentlichen Kernteilen, das die Einbettung von FPLs in ein Rechnersystem nunmehr beeinflußt. Um hier einige Beispiele zu nennen: DSP-Algorithmen, sehr schnelle Steuerungssysteme, Hardware-Beschleuniger. Die Integration von Prozessoren und FPL wird im allgemeinen als ein sehr wesentlicher Teil des Hardware/ Software Co-Designs [1] angesehen.

Dieses Gebiet erlebt seit 1991 einen großen Aufschwung, zunächst in der Forschung. Ursprünglich aus dem Ansatz stammend, das Design eines digitalen Systems nicht mehr unter dem 'Hardware First'-Ansatz durchzuführen, sondern Hardware und Software in gegenseitiger Beeinflußung und Abstimmung durchzuführen, erlebt das Co-Design durch die Verfügbarkeit von entsprechend großen FPL-Bausteinen einen erneuten Schub: Zielsystem ist sehr häufig ein Prozessor/FPL-System oder ein FPL-System selbst.

Mit diesem Ansatz wird nunmehr der Ablauf von 'Software' im Prozessor und im FPL auf eine gemeinsame Basis gestellt. Hierzu muß natürlich der Begriff der Software näher beschrieben und eingeschränkt werden. Software für einen Mikroprozessor/Mikrocontroller besteht auf der untersten Ebene aus Instruktionen, die gemäß dem von-Neumann-Modell sequentiell geladen und abgearbeitet werden. Dies wird für superskalare Prozessoren, die eine nebenläufige

Im folgenden wird eine Alternativarchitektur vorgeschlagen, die viele Aufgaben im I/O-Bereich mit keiner oder deutlich verringerter Rechenleistung des Cores durchführen kann. Diese Architektur zeichnet sich durch die Einführung strukturierbarer Hardwareelemente aus, die nach einer Initialisierung sowohl Datentransfers als auch Berechnungen durchführen können. Die hier vorgeschlagene Architektur wird als **Applikations-spezifisch strukturierbare Controller**-Architektur (***ASSC***, engl. **Application Specific Structured Controller**) bezeichnet.

Zunächst wird eine generelle Struktur auf Blockebene entworfen. Der neue Teil innerhalb dieses Blockdiagramms wird dann im Abschnitt 5.2 weiter strukturiert und anhand der im wesentlichen zu erfüllenden Aufgaben erläutert. In Abschnitt 5.3 erfolgt dann eine Erläuterung des Hardware/Software-Interfaces sowie einige Beispiele zu Anwendungen.

Der Zusammenhang mit der >S<puter-Architektur ist zunächst kaum erkennbar. Während der >S<puter auf eine Erhöhung der Performance abzielt, bietet die ASSC-Architektur eine Unterstützung des Echtzeitverhaltens. Am Schluß dieses Kapitels wird daher versucht, den Zusammenhang mit Hilfe der UCBs darzustellen.

5.1 Die ASSC-Architektur auf Blockebene

Abb. 5.2 zeigt die generelle Struktur der ASSC-Architektur. In dieser Abbildung sind ROM- und RAM-Speicherbereich als externe Speicher gezeichnet; dies ist nur exemplarisch zu betrachten, da auch speicherintegrierende Architekturen mit einem vergleichbaren, teilweise sogar verringerten Aufwand konzipiert werden können.

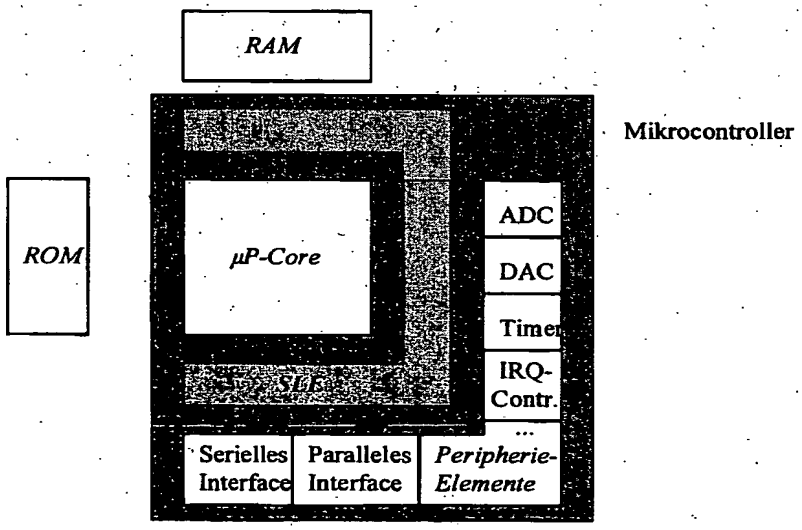

Abb. 5-2: Blockstruktur ASSC-Architektur

Im Vergleich zu einer klassischen Mikrocontrollerarchitektur wird in der ASSC-Architektur eine Schicht zwischen Peripherie-Elementen und Mikroprozessorkern gelegt, die aus strukturierbaren Logikelementen besteht und im folgenden SLE (strukturierbare Logikelemente) genannt wird. Innerhalb dieser SLE-Schicht existieren sowohl feste Verbindungen des Mikroprozessorkerns auf die Peripherieelemente, um eine Funktionskompatibilität zu bisherigen Architekturen zu schaffen, als auch konfigurierbare

Datenpfade und Datenverknüpfungen, deren detaillierte Struktur im nächsten Abschnitt und deren Kopplung zum Mikroprozessor im übernächsten Abschnitt dargestellt wird.

Die SLE-Schicht kann im Bereich des Datenspeichers in zwei Varianten aufgebaut werden:

- Variante 1 enthält keine direkte Schnittstelle zum Datenspeicher. Hierdurch wird ein konkurrierender Zugriff (Core und Peripherie via SLE) zum Datenspeicher grundsätzlich verhindert. Die SLE (und darüber die Peripherieelemente) können den Speicherzugriff indirekt durch injezierte Prozessorbefehle durchführen.
- Variante 2 enthält eine direkte Schnittstelle zum Datenspeicher (in Abb. 5-2 gestrichelt angedeutet) und muß daher auch, um konkurrierende Zugriffe lösen zu können, eine erweiterte Busschnittstelle anbieten. Der Vorteil dieser Lösung liegt in der wesentlich schnelleren Bedienung der Speichertransfers bei allerdings erhöhtem Aufwand.

Beiden Varianten gemeinsam sind die übrigen Eigenschaften der SLE-Schicht, die ein intelligentes, konfigurierbares Interface zwischen Peripherie und Core sowie den Peripherieelementen untereinander bildet.

5.2 Die Feinstruktur der strukturierbaren Logikelemente

Prinzipiell ist eine einheitliche Blockstruktur in der SLE-Schicht möglich. Diese besteht dann aus ein oder zwei (hintereinander liegenden) Logikblöcken für disjunktive Normalform (oder anderen Basissystemen), so daß (nahezu) beliebige Kombinationen der Eingangssignale erzeugt werden können.

Diese Architektur ist sehr flexibel, erzeugt jedoch eine große interne Verschaltungskomplexität, da jeder Eingang und Ausgang jeweils zweifach an die Logikgatter geführt werden muß. Im Rahmen der DNF-Architektur entstehen hierdurch AND-Gatter mit sehr vielen Eingängen. Aus diesem Grund wird im Rahmen dieses Papers eine weitere Sub-Blockstruktur vorgeschlagen, die die Komplexität der Gatter mindert; bei entsprechender Wahl der Aufteilung sowie Querverbindungen zwishen den Blöcken wird die Flexibilität der ASSC-Architektur hierdurch kaum vemindert.

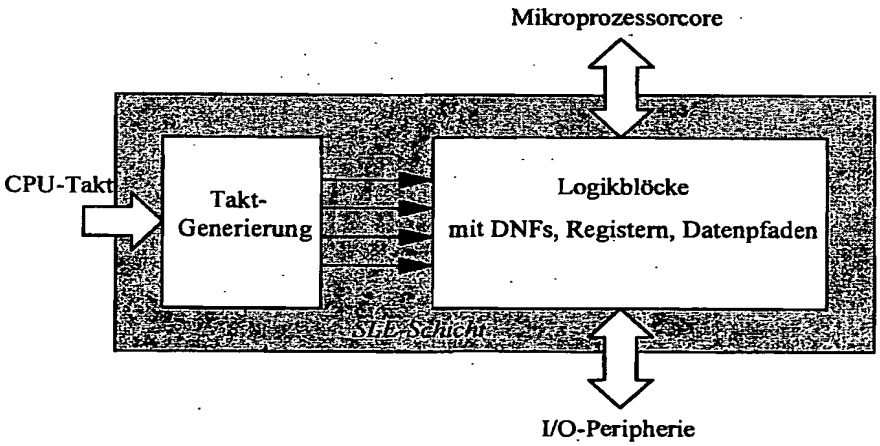

Abb. 5-3: Interne Struktur der SLE-Schicht

Die Feinstruktur der SLE-Schicht weist hierzu zunächst zwei interne Blöcke aus, die in Abb. 5-3 dargestellt sind. Der wesentlich kleinere Block dient der Versorgung der strukturierbaren

Hardware mit verschiedenen Takten, hergeleitet aus einem CPU-Mastertakt. In dieser Subeinheit werden beispielsweise Takte mit niedrigerer Frequenz, unterschiedlichem Tastverhältnis und ggf. verschobener Phasenlage generiert.

Der wesentlich größere Teil der SLE-Schicht, die eigentlichen Logikblöcke, enthält mit Hilfe von Multiplexern konfigurierbare Datenpfade, Register sowie strukturierbare Logik in Form von disjunktiven Normalformen; Multiplexer und strukturierbare Logik können dabei zusammengefaßt werden. Hier werden einzelne Zustandsautomaten mit entsprechenden Decodierungen implementiert, die dann die Kopplung zwischen Core und Peripherie sowie zwischen Peripherie/Peripherie und Peripherie/Speicher bestimmen.

5.2.1 Die Teileinheit zur Taktgenerierung

Die Teileinheit zur Taktgenerierung besteht aus einer schnellen, konfigurierbaren Logik, beispielsweise in Form einer disjunktiven Normalform (NICHT-UND-ODER-Logik) mit nachgeschalteter, konfigurierbarer Speicherung einschließlich Invertierung/Pufferung und Rückkopplung. Diese Teileinheit besitzt als Takteingang einen, ggf. mehrere CPU-Takte sowie eine Anzahl von k Ausgängen, die alle innerhalb der Logikblockteileinheit als Takt verwendet werden können. Abbildung 5-4 zeigt den Aufbau auf Gatterebene; diese Darstellung bezieht sich auf die bei PALs/GALs übliche Zeichnungsweise und enthält die prinzipielle Beschaltung von je einem internen Signal und einem generierten Ausgangstakt.

Die Ausführung als DNF (mit konfigurierbarer Invertierung) gestattet hierbei die Erzeugung einer Vielzahl von Taktvariationen; interne Register/DNFs, die rückgekoppelt sind, sorgen auch für Variationen mit nicht-binären Tast- und Teilungsverhältnissen.

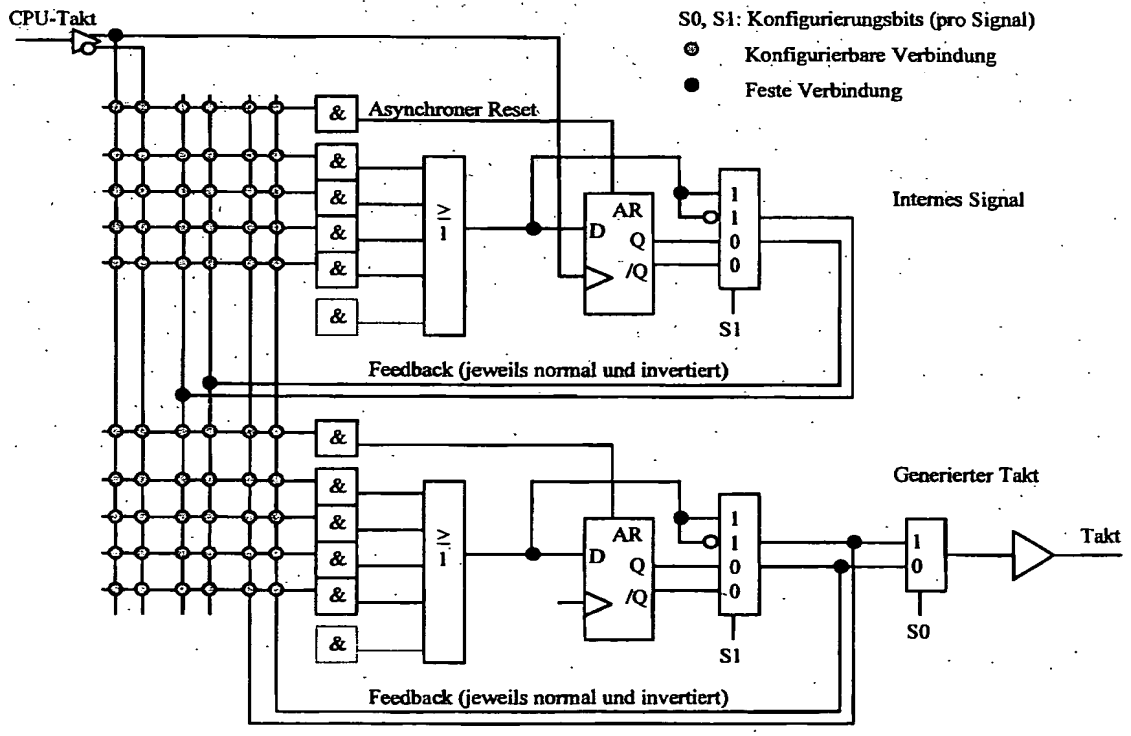

Abb. 5-4: Aufbau Taktgenerierungseinheit auf Gatterebene

Die Anzahl der erzeugbaren unterschiedlichen Takte sollte 4 nicht unterschreiten, während nach oben eine Grenze durch die Ressourcen gegeben ist. Die untere Grenze für die Anzahl der Terme pro DNF beträgt ebenfalls 4. Pro generiertem Takt sollte ein internes Signal zusätzlich zur Verfügung stehen.

5.2.2 Die Teileinheit der Logikblöcke

Wie bereits kurz dargestellt, besteht die Teileinheit der Logikblöcke aus verschiedenen Komponenten. Hierzu zählen:

1. Konfigurierbare Multiplexer zur Schaltung von Datenpfaden; diese Multiplexer fallen ggf. mit der unter 4. genannten strukturierbaren Hardware zusammen
2. Register in entsprechender Bitbreite zur Zwischenspeicherung von Daten
3. Bit-Register zur Codierung von Zuständen
4. Strukturierbare Hardware zur Decodierung von Zuständen für Steuerungssignale
5. Feste sowie strukturierbare Hardware zur Verknüpfung von Daten miteinander und mit Konstanten

Diese Bestandteile werden in folgender, in Abb. 5-5 dargestellten Weise zu einem oder mehreren Logikblöcken gekoppelt.

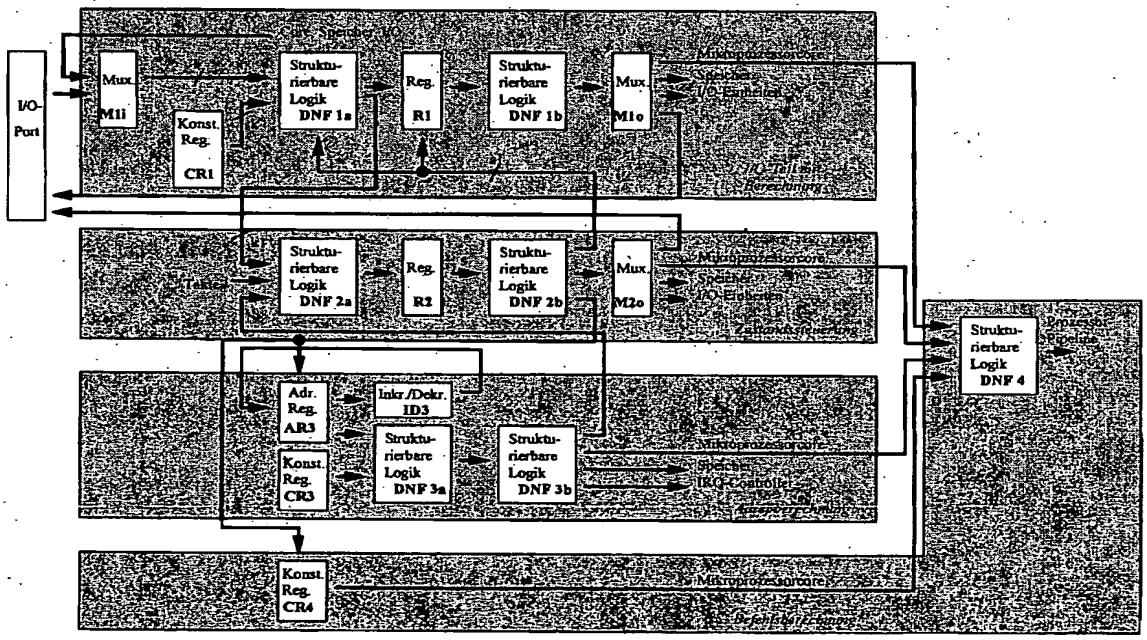

Abb. 5-5: Darstellung eines Logikblocks als Teileinheit in der SLE-Schicht

Im I/O-Teil eines Logikblocks können die Daten miteinander verknüpft werden; dies ist in konfigurierbarer oder fester Hardware möglich, in Abb. 5-5 werden die Verknüpfungen in den DNF 1a und 1b integriert. Die Speicherfunktion in R1 ermöglicht ein Zwischenspeichern, während im Konstantenregister CR1 beispielsweise Berechnungskonstanten gespeichert werden (z.B. zur Minimum/Maximum-Begrenzung).

Die Multiplexer M1i (Quellenauswahl) und M1o (Zielauswahl) verbinden den Logikblock mit Datenbussen zum Core, Speicher (falls dies zulässig ist, z.B. in Variante 2) und I/O-Elementen. Die Konstante k bezieht sich auf die systemimmanente Datenbusweite, z.B. 16 Bit. Der Multiplexer M1i ist dabei entscheidend für die flexible Kopplung zwischen den Blöcken; so ist die gemeinsame Berechnung in mehreren Blöcken für eine I/O-Komponente möglich, falls die Eingangsmultiplexer die entsprechenden Eingangsbusse schalten können. Dieses Verfahren liefert sowohl Fähigkeiten zur parallelen wie seriellen Nutzung von Berechnungsmöglichkeiten.

Die in Subblock 1 integrierte Funktionalität dient dementsprechend dem Datenfluß, diese Teileinheit ist mit der ALU einer von-Neumann-CPU vergleichbar, wobei auf die prinzipiellen Unterschiede zwischen Prozessor und SLE-Schicht an dieser Stelle verwiesen wird.

Die Zustandssteuerung benötigt einen separaten Logikteil, ausgestattet mit DNF 2a, 2b, Registern R2 und dem Multiplexer M2 zur Zielauswahl. In dem Logikteil lassen sich verschiedene Automatentypen (bis zu Mealy) integrieren, auch mit verschiedenen Takten. In der Regel wird die Zahl der Zustände klein sein, so daß bei (s =) 5 Bitregistern (32 Zustände) eine ausreichende Anzahl vorhanden ist. DNF 2b dient der Zustandsdecodierung für die Steuerung.

Die Teileinheit 2 ist in ihrer Aufgabe mit dem Steuerwerk einer von-Neumann-CPU vergleichbar, wobei die Bearbeitung von Befehlen naturgemäß entfällt; die Zustände werden durch äußere Signale getriggert durchlaufen.

Die Adreßberechnung ist optional und findet insbesondere bei blockweisen Datentransfers Anwendung. Hierfür ist ein Adreßinkrement/-dekrement sowie eine Kopplung zur Ablaufsteuerung integriert. Die Kopplung zum Mikroprozessorcore und zum Speicher soll dabei andeuten, daß in diesem Fall die Quell- bzw. Zieladresse anzugeben ist. Auch hier entfällt z.B. die Speicherkopplung im Fall der Variante 1 (Entfall des direkten Speicherzugriffs). CR3 enthält die Endadresse, in AR3 wird die aktuelle Adresse gespeichert. Die Trennung der Logik in DNF 3a und 3b soll andeuten, daß hier ein zweistufiges DNF-Konzept vorliegen sollte, da beispielsweise die Vergleichslogik für einstufige Verknüpfung sehr komplex wird.

Die Befehlsberechnung schließlich dient der Injektion von Mikroprozessorbefehlen in die Pipeline des Cores. In CR4 wird der entsprechende Befehlscode gespeichert, um diesen zusammen mit Adreß- und Dateninformationen übermitteln zu können. Die Steuerung der Injektion von Befehlen erfolgt durch Signale aus der Zustandssteuerung. Die an den Multiplexern gezeigten Verbindungen zum Mikroprozessorcore werden zu diesem Zweck in einem Interface DNF 4 zusammengefaßt, das bei entsprechender Steuerung durch den Zustandsteil einen Befehl injiziert.

Die strukturierbare Logik kann in einer beliebigen Form integriert werden; denkbar sind hier NAND-Arrays, Multiplexer-basierte Varianten und die Logik in Form der disjunktiven Normalform, die beispielhaft hier gewählt wurde. Derartige DNF-basierte Logik ist in Abb. 5-6 dargestellt (siehe auch Taktteileinheit, eine Variation der unten aufgeführten Darstellung):

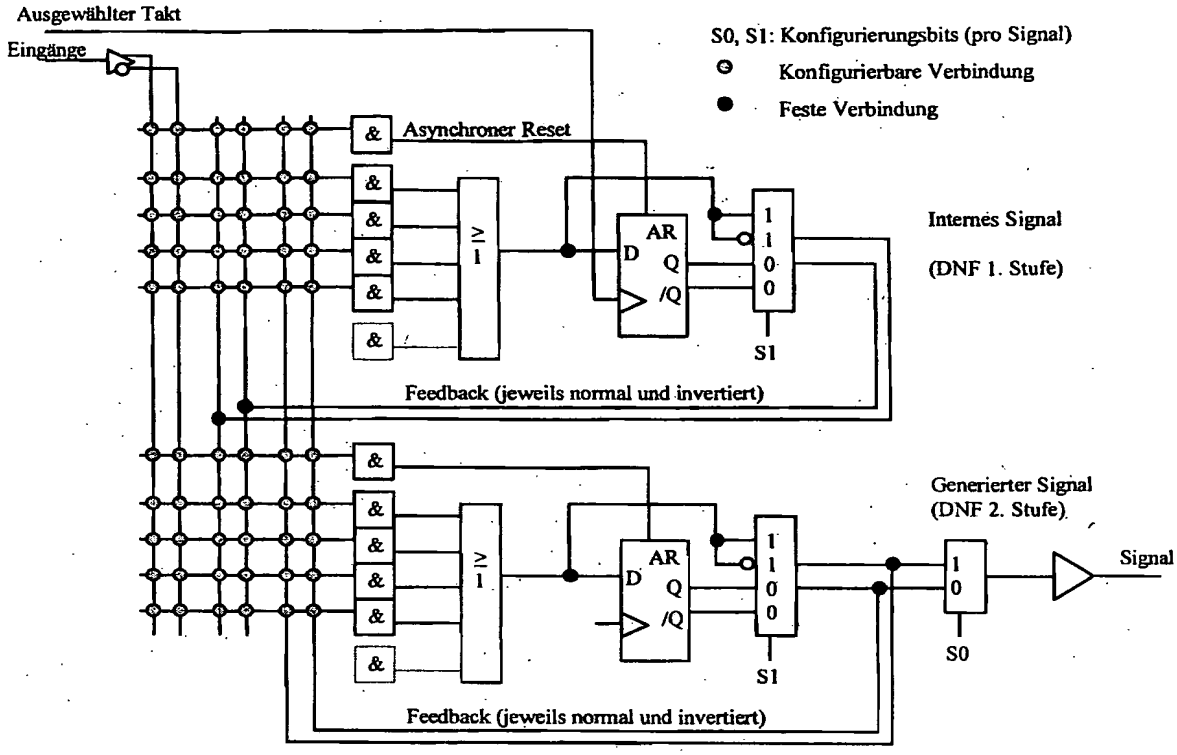

Abb. 5-6: Allgemeine Struktur der DNF in Abb. 5-5

Die hier dargestellte Form der DNF ermöglicht zweistufige Logiken, also beispielsweise die Zusammenfassung von DNF2a, R2 und DNF2b mit zusätzlichen Ausgangsregistern; dies kann jederzeit zu einer einstufigen Logik durch Fortlassen der 2. Stufe und Integration des Ausgangsmultiplexers in Stufe 1 verringert werden. Die Ausdehnung der jeweiligen Logikstruktur muß durch den Systemdesigner festgelegt werden, sie ist charakteristisch für die Flexibilität und Leistungsfähigkeit der ASSC-Architektur.

Abschließend ist zu bemerken, daß bei Datengruppenoperationen zusätzliche Speichereinheiten in den SLE zu integrieren sind. In diesem Fall müssen auch vergangene Daten in die Berechnung eingehen, deren Speicherung aus Performancegründen nicht im Speicher, sondern im SLE-Teil selbst vorgenommen werden sollte.

5.3 Das Hardware-Software-Interface

In den konfigurierbaren Elementen der SLE-Schicht – hierzu zählen die Multiplexer, die Verbindungen innerhalb der DNF und die Konfiguration der Register – müssen Informationen ablegbar sein, die das Verhalten der Schaltung repräsentieren. Die Technologie der Informationsspeicherung ist prinzipiell auf drei Wegen vorstellbar:

- Es werden einmalig herstellbare oder löschbare Verbindungen genutzt, die unter den Bezeichnungen Fuse bzw. Antifuse bekannt sind. Diese Verbindungen können sowohl fabrikatorisch als auch durch geeignete Programmiergeräte im Feld gesetzt werden.
- Zur Informationsspeicherung wird eine EPROM- bzw. EEPROM-basierte Technologie genutzt. Diese Speicherung ist prinzipiell lösch- und wiederbeschreibbar, sie erfolgt durch

This Page is Inserted by IFW Indexing and Scanning Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☑ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☐ BLURRED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLOR OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☑ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REFERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.
As rescanning these documents will not correct the image problems checked, please do not report these problems to the IFW Image Problem Mailbox.